


5

-1-

Certificate of Mailing Under 37 C.F.R. 1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as Express Mail in an envelope addressed to: **BOX PATENT APPLICATION**, Assistant Commissioner for Patents, Washington, DC 20231 on:

Date: September 28, 2001 Express Mailing Label No.: EF384081185US

Signature: 

Typed or Printed Name: Crystal Slason

Inventors: Richard Francis Cormier, Andrew Bruce and  
Svetlana Patsenker  
Attorney Docket No.: EMC01-11(01046)

**METHODS AND APPARATUS FOR MANAGING PLUG-IN SERVICES**

**FIELD OF THE INVENTION**

The present invention generally relates to computer and software environments that require the use of services, and more particularly, to systems and techniques that manage plug-in services on behalf of software applications and processes requiring access to plug-in services.

**BACKGROUND OF THE INVENTION**

Many types of conventional computing systems operate software programs that require services of various types during their operation. By way of example, a software application such as an Internet web browser which operates (e.g., executes) as a web browser process within a computer system may require audio and/or video playback services, document rendering services, document interpretation services or other services for which service logic (i.e., software code or instructions that provide such services) is

not built-in to the web browser application itself. The reason that developers of software applications do not often provide all required service logic for every service within the application code is because service software developers are constantly developing new services that applications may require access to for various purposes. In addition, service capabilities can change when service developers update service logic to accommodate new features or revisions. For these reasons, many conventional software applications access services by incorporating the use of software modules called “plug-ins.”

Generally, a conventional plug-in module is a software procedure, library routine, function, or other entity which a service developer creates to operate in a computer system in order to provide required processing functionality for a particular purpose. The purpose of the plug-in module is referred to as the “service” provided by the plug-in module. A typical plug-in module incorporates the use of a standardized application programming interface (API) that allows other software applications, such as the web browser process mentioned above, to access the service provided by the plug-in module by making a function or system call to the service using the plug-in application programming interface. In operation, the application programming interface for a plug-in module offering a service receives a set of input variable values which the service logic and processing within the plug-in module can operate upon in order to provide the service offered by the plug-in module. The application programming interface can also return or output a result that reflects the performance of the service logic upon the input variable values. A plug-in module may direct the output, for example, to a specific output device such as the computer display or an audio or video output device.

In some instances, a conventional software application might attempt to access or process data that requires a service provided by a plug-in module that is not presently resident within the computer system in which the software application is operating. As an example, consider a situation in which a user directs a web browser to access video data that is arranged in a video format that requires the services of a video plug-in module in order for that web browser to playback the video to the user operating the web browser. Unless the web browser has previously accessed video data in this particular video format, it is unlikely that the plug-in module providing the video playback service

is resident on the computer system in which the web browser is operating. The web browser process may automatically make this determination since the video data may specify what plug-in service is required to playback the video data. The web browser can check a list of currently available plug-in services to determine if the required service is available from a resident plug-in. If the web browser process determines that the required plug-in service is not resident on the computer system, the web browser may prompt the user to obtain the required plug-in module that provides the required plug-in service, for example, via downloading the plug-in module from a plug-in developer web site. If the user instructs the web browser to download the plug-in module, the web browser can proceed to obtain (via a download process), install and operate (i.e., run, interpret or otherwise perform or access) the required plug-in module in order to obtain and use the video playback service that the plug-in module provides.

#### SUMMARY OF THE INVENTION

Conventional techniques for accessing plug-in modules and plug-in services suffer from a number of deficiencies. By way of example, if a particular plug-in module provides a plug-in service that operates using a proprietary application programming interface, a software application that desires to use that plug-in service must contain logic instructions and/or function calls which conform or adhere to the proprietary application programming interface requirements. As a specific example, if a conventional plug-in module provides a proprietary naming service, applications that desire to use that naming service must include service calls to the plug-in module that conform to the proprietary naming service application programming interface. Once a software developer creates a software application that contains plug-in service calls that conform to the proprietary format, if the user of the software application later decides to replace the naming service with a new naming service provided by a plug-in module created by a new software developer, the plug-in service calls within the software application that conform to the old proprietary format will no longer be usable with the new plug-in service. As a result, the developer of the software application must make modifications to the plug-in service

calls in the application to conform to the new format required by the new naming service plug-in.

Another drawback that is present in conventional techniques and mechanisms for managing plug-in services relates to difficulty in determining when a plug-in service is properly available for use by a particular software application. As an example, in a complex computing system environment, it may be the case that one plug-in service relies on the use of another plug-in service for certain operational capabilities. This is called a plug-in or service “dependency.” It may be the case, for example, that a video playback plug-in service also requires the use of an audio playback plug-in service in order for audio data, which is encoded within the video data, to be reproduced to the speakers of a computer system while the video playback plug-in service reproduces the video data on the computer display. Using this example, it may be said that the video playback plug-in module which provides the video playback plug-in service is dependent upon or requires the audio playback plug-in module that provides the audio playback plug-in service.

Continuing with this example, when a conventional software application requires the ability to playback video data (which includes playback of audio data), the software application might recognize that a required video playback plug-in module is not resident within the computer system in which the conventional software application operates. As discussed above, the conventional software application allows a user to direct the application to download the required video playback plug-in module. However, the conventional software application may be unaware that the video playback plug-in module also requires the use of an audio playback plug-in module that may not be resident within the computer system in which the conventional software application operates. As a result, upon operation of only the video playback plug-in service, the video data is either reproduced without sound, or the video playback plug-in service may be unable to playback the video at all due to an inability to operate without the use of the audio playback plug-in service.

Embodiments of the present invention significantly overcome these and other deficiencies found in conventional plug-in management and plug-in service operation

environments. In particular, embodiments of the invention provide a plug-in manager application and process that operates in a computer system to manage services associated with a plurality of plug-in modules. The plug-in manager process can operate on behalf of software applications requiring access to plug-in services by receiving, detecting or  
5 otherwise determining a list of plug-in services that the software application requires. The plug-in manager can then query a set of plug-in modules identified in the list in order to determine service dependencies between the plug-in modules. For example, if a software application indicates that three plug-in services are required during operation of that software application, the plug-in manager process can operate the plug-in modules in  
10 an initiation or enabled mode and can query those plug-in modules for a list of services that they provide and for a list services that they will require when operating in a service operation mode. Using this information, if necessary the plug-in manager can also query other plug-in modules which were not identified by the software application, but which supply or provide the services that are required by the plug-in modules that the software  
15 application did identify. This process can continue until the plug-in manager discovers and queries all plug-in modules.

Once the plug-in manager obtains the dependencies of all plug-in modules, the plug-in manager can calculate a plug-in initiation sequence or order based upon the dependencies identified between the plug-in modules. This may be done, for example, by  
20 analyzing the plug-in module dependencies to produce a dependency tree that contains nodes that represent plug-in modules and whose hierarchy represents a plug-in module initiation order. For example, the lowest or leaf nodes in the plug-in module dependency tree identify services of plug-in modules that should be initiated or started first whereas upper-level nodes in the tree identify plug-in modules that provide services that should be  
25 initiated after their child nodes in the dependency tree.

Using the calculated plug-in initiation order, the plug-in manager can initiate service operation of the plug-in modules according to the plug-in initiation order. As a result, if a first plug-in module provides a service required by a second plug-in module, the first plug-in module will be initiated such that the service provided by the first plug-in  
30 module is available to the second plug-in module when required by the second plug-in

module. That can result, for example, in plug-in modules that provide services that other plug-in modules rely upon being started prior to the plug-in modules that require those services. In this manner, embodiments of the invention help to insure that plug-in modules are initiated in a proper sequence so that all services required by plug-in modules will be available when the plug-in modules requiring those services are initiated.

In some instances, if two plug-in modules rely on the services of each other, the initiation order may be to start the services of the plug-in modules concurrently or, one before the other. In such a situation, one plug-in module can include a wait state operation that the plug-in manager can operate to cause the plug-in module to wait to provide its service until the other plug-in module begins providing service. Alternatively, the wait state operation can cause one plug-in module to signal to the other plug-in module (the other one that is dependent upon the one providing the signal) to indicate that a service is not available. This allows two plug-in modules that depend upon each other to signal to each other to allow one to start followed by the other in an order which the plug-in modules can determine to avoid a deadlock in the plug-in module service initiation order.

In addition, certain embodiments of the invention operate using Java or other object-oriented interfaces within the computer system environment for communication between the plug-in manager and the plug-in modules. Using such interfaces avoids having to incorporate different proprietary code for multiple interfaces to different proprietary plug-in modules.

Generally, one method embodiment of the invention operates to manage plug-in services by obtaining identities of a plurality of plug-in modules and retrieving a dependency list indicating respective plug-in services provided by, and required by, each plug-in module identified in the identities of a plurality of plug-in modules. As noted above, a plug-in manager of performing this method is thus able to determine what services are provided and required by all plug-in modules. Thereafter, the method calculates a plug-in initiation order based upon the dependency list indicating respective plug-in services provided by, and required by, each plug-in module. The method then initiates service operation of plug-in modules according to the plug-in initiation order,

such that if a first plug-in module provides a service required by a second plug-in module, the first plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required by the second plug-in module.

5 Other embodiments of the invention include a computerized device, such as a workstation, computer system or other such device configured to process all of the method operations disclosed herein as embodiments of the invention. In such embodiments, the computer system includes an interface, a memory, a processor and an interconnection mechanism connecting the interface, the processor and the memory. The  
10 memory is encoded with a plug-in manager application that when performed on the processor, produces a plug-in manager process that manages services associated with a plurality of plug-in modules by causing the computer system to perform any or all of the method embodiments and operations explained herein as embodiments of the invention. In other words, a computer system or computerized device programmed or configured to  
15 operate according to the method operations explained herein is considered an embodiment of the invention.

Other arrangements of embodiments of the invention that are disclosed herein include software programs to perform the method embodiment operations summarized above and disclosed in detail below. More particularly, a computer program product is  
20 disclosed which has a computer-readable medium including computer program logic encoded to provide management of plug-in services and plug-in modules as explained herein. The computer program logic, when executed on at least one processor within a computer system, causes the processor to perform the operations (e.g., the methods) indicated herein as embodiments of the invention. Such arrangements of the invention  
25 are typically implemented as software, code and/or other data arranged or encoded on any type of computer readable medium such as a magnetic or optical medium (e.g., CD-ROM), floppy or hard disk, or other a storage medium such as a memory or memory system, firmware or microcode in one or more Read Only Memory (ROM) or Random Access Memory (RAM) chips or other microchips or as an Application Specific  
30 Integrated Circuit (ASIC) or any equivalents thereof. The software, firmware or other

such computer readable medium configuration can be installed and operated within a computer system to cause the computer system to perform the techniques explained herein as embodiments of the invention. It is to be understood however that such embodiments may comprise the computer readable medium encoded with the data and instructions to carry out the processing of the method embodiments explained herein, and that the computer readable medium does not need to be installed, coupled or otherwise configured (i.e., loaded and read) within a computer system that can read the computer readable medium. The encoded medium alone is thus an embodiment.

It is also to be understood that the system of the invention can be embodied strictly as a software program, as software and hardware, or as hardware alone (e.g., via implementation in circuitry). Example embodiments of the invention may be implemented within the software applications that provide and manage plug-in functionality in a storage area network (SAN) computing system environment such as the EMC Control Center Software application manufactured by EMC Corporation of Hopkinton, Massachusetts, USA.

#### BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular description of preferred embodiments of the invention, as illustrated in the accompanying drawings in which like reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale, with emphasis instead being placed upon illustrating the embodiments, principles and concepts of the invention.

Figure 1 illustrates an example of a computing system environment including a computer system operating a plug-in manager process configured according to one embodiment of the invention.

Figure 2 is a flow chart of processing steps as performed by embodiments of the invention to manage initiation of plug-in services in a computing system environment.



Figure 3 illustrates an example architecture of the computer system operating a plug-in manager application and process configured according to one embodiment of the invention.

Figure 4 illustrates a software and interface architecture for a plug-in module configured in accordance with one embodiment of the invention.

Figure 5 illustrates a software architecture of a plug-in manager process and communications that can take place between the plug-in manager process and a server process and a plurality of plug-in modules in accordance with one example embodiment of the invention.

Figures 6 and 7 provide a flow chart of processing steps that illustrate an example operation of the computer system configured in accordance with one embodiment of the invention.

Figure 8 illustrates an example of a plug-in initiation order in the form of a plug-in module dependency tree.

## DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

Generally, embodiments of the invention provide unique mechanisms and techniques that operate to provide a plug-in manager that manages plug-in services and plug-in module service invocation (e.g., startup and shutdown) on behalf of software applications requiring access to the plug-in services. As an example, when a software application such as a client or server process requires access to one or more plug-in modules for use of particular plug-in services offered by those plug-in modules, a plug-in manager configured according to embodiments of the invention is able to obtain the identity of the plug-in modules for which access is requested. The plug-in manager can then instantiate each plug-in module into an enabled state and can query a dependency interface associated with each plug-in module in order to determine plug-in dependencies associated with that plug-in module. In response to a dependency query, a plug-in module can provide, to the plug-in manager, a list of services provided by that plug-in module as well as a list of services required by that plug-in module. By querying all plug-in modules identified by a software application (and any plug-in modules which

those plug-in modules require), the plug-in manager can determine all services that are required by or provided by the complete set of plug-in modules that will need to operate to provide the required services requested by the application.

Using this information, the plug-in manager configured according to  
5   embodiments of the invention can further operate to calculate a plug-in startup or initiation order that is used to initiate plug-in modules into a started state during which those plug-in modules provide their respective services. The plug-in manager can then initiate service operation of the plug-in modules according to the initiation order. As a result of initiating plug-in modules according to the calculated initiation order, the plug-  
10   in manager can help to ensure that plug-in modules that require services offered by other plug-in modules are not prematurely started. In addition, if two or more plug-in modules require the services of each other for proper operation (e.g., they are interdependent upon each other), the plug-in manager can initiate each plug-in module at the same approximate time (or one before the other) and each interdependent plug-in module can  
15   operate a wait-state operation which allows them to signal between each other at an appropriate time to begin service operation to avoid service deadlocks.

A plug-in manager configured according to embodiments of the invention can also operate to publish a list of services available for the current set of plug-in modules which are operating in a started state (i.e., that can provide services). This published list  
20   of services is accessible by software applications that require plug-in services thus allowing the applications to quickly identify and access plug-in module services which they may require during their operation.

Figure 1 illustrates an example of a computing system environment 100 suitable for use in explaining example operations of embodiments of the invention. The  
25   computing system environment 100 includes a network 105 which may be any type of data or communications network arranged in a local area network (LAN), wide area network (WAN), Internet or storage area network (SAN) configuration (as in this example) or any combination thereof. The network 105 couples storage area network resources 109, a client computer system 110 and a server computer system 120. The  
30   network 105 allows communications to take place between the storage area network

resources 109 and the client and server computer systems 110 and 120. The client and server computer systems 110 and 120 can be any type of computing system such as a workstation, personal computer, mainframe, portable computing device or the like. The server computer system 120 in this example includes a coupling to a plug-in database 155 that contains plug-in module definitions 162 and plug-in initiation information 164.

The client computer system 110 operates a client process 115 that, in this example, is a storage area network (SAN) manager console process that allows a user 108 to manage storage area network resources 109 operating within the storage area network 105. The storage area network resources 109 can be any type of manageable hardware or software entities such as data storage systems, storage area network data communication devices (e.g., SAN switches), and/or storage area network software applications operating in such devices. The client process 115 intercommunicates over the network 105 with a server process 145 operating within the server computer system 120. In this example, the server process 145 is a storage area network management server. The client process 115 accesses (e.g., exchanges commands with) the server process 145 in order to obtain management data concerning the storage area network resources 109 and to direct the server process 145 to carry out storage area network management operations (i.e., management commands) upon the storage area network resources 109. During such processing, either the client process 115 and/or the server process 145 may require the use of plug-in services provided by plug-in modules 160-1 through 160-N operating within the server computer system 120.

Besides operating the server process 145, in this example, the server computer system 120 further operates (i.e., executes, runs, interprets or otherwise performs) other software processes including a plug-in manager process 150 configured according to embodiments of the invention and one or more (a plurality in this example) plug-in modules 160-1 through 160-N. Each plug-in module 160 can operate to provide one or more services 147 to software applications such as the client 115 and/or server processes 145 that require such services (or to other entities such as hardware devices that may require services 147). The processing and service logic of each plug-in module 160 is defined within the plug-in module definitions 162 which may be, for example, Java or

other object-oriented classes that may be instantiated (as will be explained) to operate as the plug-in modules 160 within the server computer system 120.

In this example, the plug-in manager process 150 operates according to embodiments of the invention to “manage” the plug-in modules 160. Generally, what is meant by the term “manage” is that the plug-in manager process 150 can invoke or instantiate operation plug-in module definitions 162 as plug-in modules 160, and can control the state of the plug-in modules 160, including when they are instantiated into an initiation or enabled state during which time they are performing (e.g., executing or are being interpreted) but do not provide services, and when they are transitioned from the enabled state to a started state during which time they provide their respective services 147. The plug-in manager process 150 can also manage the plug-in modules 160 to transition them from the started or service-providing state back to the enabled state and can, if required, terminate operation of the plug-in modules 160. In addition to managing their run-time operational state, the plug-in manager process 150 can manage the order in which plug-in modules are initiated and terminated in relation to each other such that plug-in module dependencies are not violated. The plug-in manager process 150 can also create and maintain a published list of services 180 indicating which plug-in modules 160 are currently operating and available to provide which respective services 147.

Figure 2 is a flow chart of processing steps which the server computer system 120 configured to operate the plug-in manager process 150 performs in accordance with one embodiment of the invention. The processing steps in Figure 2 will be explained with reference to the example computing system environment 100 illustrated in Figure 1 in order to provide an example of the general operation of embodiments of the invention.

In step 200, a process requiring plug-in services 147 is initiated. Such a process may be a hardware or software entity operating anywhere within the computing system environment 100. With respect to the example in Figure 1, the client process 115 and the server process 145 interoperate under the control of the user 108 to provide management functionality of storage area network resources 109. During this processing, the client process 115 and/or server process 145 may require access to service functionality

provided one or more of the plug-in modules 160. In this particular example, the server process 145 determines that it requires access to plug-in services 147.

Next, in step 201, the process requiring plug-in services (i.e., the server process 145 in the example of Figure 1) provides a request containing a list of services 170 (i.e., plug-in services that are required) to the plug-in manager process 150. The list of services 170 contains the identity of one or more specific plug-in services 147. The server process 145 (or the client process 115) may forward the list of plug-in services 170 in a service request or system call to an operating system performing within the server computer system 120. The plug-in manager process 150 may operate in conjunction with such an operating system in order to intercept or otherwise retrieve the list of plug-in services 170.

In step 202, the plug-in manager 150 obtains the identities of a plurality of plug-in modules 160 based upon the plug-in services identified in the list of services 170. As an example, the names of the particular plug-in services may correspond to the identity of plug-in module definitions 162 defined within the plug-in database 155.

In step 203, the plug-in manager retrieves a dependency list indicating respective plug-in services provided by, and required by, each plug-in module 160 identified in the identities of the plurality of plug-in modules (obtained in step 202). To perform step 203, the plug-in manager process 150 can access the plug-in module definitions 162 (e.g., Java classes) to instantiate respective plug-in modules 160 corresponding to the services identified in the request 170. The plug-in modules 160 are instantiated into an enabled state in which each plug-in module 160 is resident in memory within the server computer system 120 but is not yet providing its respective service 147. Once each plug-in module 160 is operating in the enabled state, the plug-in manager 150 can provide a dependency query 171 to each plug-in module 160 to determine what services that plug-in module 160 provides and to also determine what services that plug-in module 160 may require to properly operate. In response to such a query, each plug-in module 160 can provide a dependency response 172 back to the plug-in manager process 150 indicating services provided and services required. The collective set of services provided by and required

for each respective plug-in module 160 comprises a dependency list compiled, retrieved or otherwise developed by the plug-in manager process 150.

Next, in step 204, the plug-in manager process 150 calculates a plug-in initiation order based upon the dependency list that indicates the respective plug-in services 147 provided by and required by each plug-in module 160. In other words, in step 204, the plug-in manager process 150 figures out an order in which plug-in modules 160 must be transitioned from the enabled state into a started state in which they can provide their respective services, such that plug-in modules that depend upon or require the services 147 of other plug-in modules are started after the plug-in modules 160 providing those required services.

Next, in step 205, the plug-in manager process 150 initiates service operation of plug-in modules 160 according to the plug-in initiation order calculated according to the processing performed in step 204. The plug-in manager process 150 initiates the service operation 147 of the plug-in modules 160 in step 205 such that if a first plug-in module, such as plug-in module 160-1, provides a service (e.g., 147-1, not specifically labeled) that is required by a second plug-in module 160-2, the first plug-in module 160-1 is initiated such that the service 147-1 is available to the second plug-in module 160-2 when required by the second plug-in module 160-2. Generally, in such a case, the first plug-in module 161 is initiated into the started or “service operation” state before the second plug-in module 160-2. The plug-in manager 150 can initiate service operation in a plug-in module 160 by sending a “dependency available” message 173 to that plug-in module 160. The dependency available message 173 indicates the services that are now available for use by that particular plug-in module 160. These may be, for example, the services that that particular plug-in module 160 requires to properly operate.

The dependency available message 173 thus serves as a “go-ahead” message indicating to the plug-in module 160 that a set of services are now available for use by the plug-in module 160. In response to receiving a dependency available message 173, a plug-in module 160 begins to provide a service 147 and replies to the plug-in manager process 150 with a “services available” message 174 which indicates what services 147 that plug-in module 160 is currently able to provide. Generally, the services available

message 174 identifies the services that the plug-in module 160 indicated that it could provide in the dependency response message 172. However, in situations where to plug-in modules 160 depend upon each other, it may be the case that the services available message 174 indicates only a subset of all services that the plug-in module 160 is capable of providing and that at a later time, the plug-in module 160 will provide another services available message 174 back to the plug-in manager process 150 to indicate that additional services are now also available.

In this manner, the plug-in manager process 150 configured according to embodiments of the invention helps to ensure that plug-in modules 160 begin providing services 147 in an order which avoids a service deadlock situation in which one plug-in 160 cannot properly operate since a service it requires is not available.

Before explaining additional operations of embodiments of the invention, further details of the architecture and inter-operation between the plug-in manager process 150 and plug-in modules 160 will be explained.

Figure 3 illustrates an example architecture of a server computer system 120 configured in accordance with one embodiment of the invention. The server computer system 120 can be any type of computerized device such as a personal or portable computer (e.g., laptop, handheld computing device or desktop PC), workstation, minicomputer, mainframe or the like. The server computer system 120 includes an interconnection mechanism 121 that couples a memory 122, a processor 123, and an interface 124. The memory 122 can be any type of computer readable medium such as a semiconductor memory (random access or read only), or may be a volatile memory or a storage medium such as a disk, tape or other medium.

In this example, the memory 122 is encoded with a server application 146, a plug-in manager application 151, one or more plug-in modules 160 and a published list of services 180 that represent data and/or logic instructions (e.g., software code or data in memory) that embody the data structures and processing functionality associated with embodiments of the invention. The processor 123 can access the memory 122 via the interconnection mechanism 121 in order to run, execute, operate, interpret or otherwise access and/or perform the data and logic instructions of the server application 146, the

plug-in manager application 151 and plug-in modules 160 to produce a respective server process 145, plug-in manager process 150 and executing (or interpreted) plug-in modules 160. For example, the plug-in manager process 150 represents the operation or execution of one or more portions of the plug-in manager application 151 by the processor 122 in the server computer system 120.

The published list of services 180 within the memory 122 represents a data structure, database or other encoding of data that contains the identity of services 147 and/or plug-in modules 160 that are presently operating in the service operation state to provide services 147 to process such as the server process 145.

Is to be understood that embodiments of the invention include the plug-in manager application 151 (i.e., the un-executed or non-performing logic instructions and/or data) encoded or residing within any type of computer readable medium such as a floppy disk, hard disk or optical medium, or in a memory system such as in firmware, read only memory (ROM), or, as in this example, as executable code within the memory system 122 (e.g., within random access memory or RAM). It is also to be understood that other embodiments of the invention comprise the plug-in manager process 150 operating or performing within the processor 123 in the server computer system 120. While not shown in this example, those skilled in the art will understand that the computer system 120 may include other processes and/or software and hardware components, such as an operating system, which have been omitted from this illustration for ease of description of the invention.

Figure 4 illustrates an example of a software architecture for a plug-in module 160 configured according to one embodiment of the invention. The plug-in module 160 includes a set of “service provided” interfaces 261 (or just service interfaces 261) which in this example include plug-in service “A” interface 261-1, plug-in service “XYZ” interface 261-2 and any other plug-in service interfaces 261-M. There may be one or more of such service interfaces 261 for each plug-in module 160. Each plug-in service interface 261 represents a service 147 that the plug-in module 160 can provide. In this example then, the plug-in module 160 illustrated in Figure 4 provides the two services (generally, services shown by 147 in Figure 1) named “A” and “XYZ.”



The example plug-in module 160 in Figure 4 further includes a list of required services 162, which in this example include services “C,” “D” and “E”. The list of required services 162 indicates which services this particular plug-in module 160 requires during its operation in order to properly provide its own services “A” and “XYZ” defined by the services provided interfaces 261. In other words, the list of required services 162 indicates what services this plug-in module 160 relies upon to properly perform its operation.

The plug-in module 160 further includes plug-in information 165 which can include, for example, plug-in state information and plug-in property information related to the operation of the plug-in module 160. Also included are a dependency available interface 163. The dependency available interface 163 is an interface which the plug-in manager process 150 can use to transition the plug-in module 160 from an enable state to a started or service operation state which allows the plug-in module 160 to provide its services 147. The actual service processing provided by a plug-in module 160 is contained in the plug-in service logic 164 which represents any code, logic instructions and/or data that the plug-in module 160 uses during its operation to provide the services 147 defined by the various service interfaces 261.

In addition, in this example plug-in module 160, a wait state operation 165 is included which represents processing and logic instructions required to allow this plug-in module 160 to determine if other plug-in modules upon which it depends for operation are available for service access. In other words, a wait state operation 165 represents the processing functionality to allow this plug-in module 161 to signal another plug-in module that it is available for service processing or that it is waiting for that other plug-in module to provide a particular required service. As will be explained further, the wait state operation 165 is the useful in situations where the plug-in manager process 150 detects an interdependency between two plug-in modules 160. If the plug-in manager process 150 determines that two plug-in modules depend upon each other for services 147, the plug-in manager process 150 can cause each plug-in module to enter the started or service providing state and can then invoke the wait state operation 165 within one or more of these inter-dependent plug-in modules 160 to allow those plug-in modules to

intercommunicate with each other to indicate when each is available to begin providing a service upon which the other plug-in modules depend. In other words, the wait-state operation interface 165 allows the plug-in manager to pass-off the duties of coordinating a smooth (i.e., non-deadlocked) service start-up procedure between two or more plug-in modules that depend upon each other to those plug-in modules 160.

Figure 5 illustrates an example software architecture of a plug-in manager process 150 configured according to one embodiment of the invention. In addition, Figure 5 illustrates an example of communications that can take place between the plug-in manager process 150 and a server process 145 and a plurality of plug-in modules 160 in accordance with an example operation of one embodiment of the invention.

The example plug-in manager process 150 illustrated in Figure 5 includes a plug-in module identifier 151, a plug-in initializer 152, a dependency analyzer 153 and a service controller 154. In addition, the plug-in manager process 150, in this example, creates and maintains a plug-in module dependency tree 190 which serves as the plug-in initiation order as will be explained in detail below.

A plurality of plug-in modules 160-1 through 160-8 are also illustrated in Figure 5. Each plug-in module 160 is software code, logic instructions or other data or information maintained in a respective plug-in module definition 162-1 through 162-8 within the plug-in database 155 that, when instantiated in memory 121 and performed on the processor 122 (Figure 3) provides a service 147 for one or more requesting processes (e.g., client process 115 and/or server process 145). In the illustration in Figure 5, each plug-in module 160-1 through 160-8 has a respective plug-in name "A" through "G" which corresponds to the plug-in service which that plug-in module provides. As a specific example, the plug-in module 165 is named plug-in "E" and provides a service named "E" as indicated at location 161-5. In addition, each plug-in module 160 specifies any services required (i.e., depends on) for that plug-in module to properly operate. Again with respect to the example plug-in module 165, the plug-in service "E" requires or depends upon the service "C" to properly operate, as indicated at location 162-5. Some plug-in modules 160 require no other services to properly operate, such as plug-in

modules 163 (service “C”), plug-in module 164 (service “D”) and plug-in module 166 (service “F”) in this example.

Directing attention now specifically to plug-in modules 160-6 through 160-8 (services “F”, “G” and “H”), these plug-in modules provide services that include a version identification associated with the services. Specifically, these plug-in modules provide services “F.1”, “G.1” and “H.1,” where the “.1” portion of the service name indicates the version of the service that these respective particular plug-in modules 160 provide. In addition, certain plug-in modules 160 may require not only a specific service to properly operate, but may also require a particular version of the service. As an example, plug-in module 160-7 requires the services “F.1” and “H.1”. As such, if for example the plug-in module 160-8 providing the “H” service provided a service other than version “H.1” (e.g., H.3), then the plug-in module 160-8 might not provide an “H” service suitable for use by the plug-in module 160-7. Service version identification is optional.

Directing attention now in Figure 5 and specifically to the communications 170 through 177 which transpire between the plug-in manager process 150, the server process 145 and the plug-in modules 160, each communication message 170 through 177 represents a packet, command, message or other unit of data that conveys information during operation of embodiments of the invention. An example of the operation of the plug-in manager process 150 illustrated in Figure 5 will now be explained with respect to the flow chart of processing steps shown in Figure 6 through 7.

Figures 6 and 7 provide a flow chart of processing steps that illustrates the operation and interaction between the plug-in manager process 150, the server process 145 and one or more plug-in modules 160 according to one example embodiment of the invention. The processing steps shown in Figure 6 and 7 are meant to be one example of how a plug-in manager process 150 configured according to embodiments of the invention can operate to manage plug-in services. It is to be understood that variations to the order and inclusiveness of the processing steps is possible while still achieving the effects provided by embodiments of the invention to manage plug-in services in a computing system environment such as that illustrated in Figures 1 and 5. In addition, it

is to be understood that the architecture of the plug-in manager process 150 illustrated in Figure 5 is shown by way of example only to convey the various functional operations that a plug-in manager process 150 configured according to embodiments of the invention can perform.

5 In step 300, the server process 145 provides a list of services 170 to be started within the computerized device in which it operates, which in this example is the server computer system 120. As shown in the example in Figure 5, the list of services 170 identifies that the server process 145 requires the services “A,” “B,” “E,” “H.1” and “G.1.”

10 In step 301, the plug-in module identifier 151 (Figure 5) operating within the plug-in manager process 150 receives the list of services 170 to be started within the server computer system 120. This may be done, for example, by intercepting a service call or system call to an operating system performing within the server computer system 120.

15 Next, in step 302, in response to receiving a list of services 170, the plug-in module identifier 151 determines, for each service identified in list of services 170, a respective plug-in module definition 162 that can provide that service. To do so, the plug-in module identifier 151 accesses the plug-in database 155 to obtain a plug-in module definition 162 for each of the identified services contained within the list of  
20 services 170. In the illustrated example of Figure 5, the plug-in module identifier 151 obtains plug-in module definitions 162-1 (for service “A”), 162-2 (for service “B”), 162-5 (for service “E”), 162-7 (for service “G.1”) and 162-8 (for service “H.1”) based on the names of the services requested in the list of services 170.

In step 303, which is an optional step, the plug-in module identifier 151 accesses  
25 any predefined or pre-known dependencies 168 contained within the plug-in initiation information 164. Predefined dependencies information 168 can indicate other plug-in modules 160 not specified in the list of services 170 that will be required to operate the plug-in modules 160 that provide the services specified in the list of services 170. In other words, the server process 145 may provide a list of services 170 for certain plug-in  
30 modules but may be unaware of other plug-in modules 160 which will services may

require to operate. The predefined dependency information 168 can indicate any other plug-in modules 160 which are required to operate the plug-in modules 160 identified with respective services listed in the list of services 170 (but which are unknown to the server process 145). Once the plug-in module identifier 151 has determined all plug-in module definitions that will be required to operate, the plug-in module identifier places the identity of each plug-in module definition into a set of identities of plug-in modules 191 and passes the set of identities of plug-in modules 191 to the plug-in initializer 152.

In an alternative configuration, the plug-in module identifier 151 can simply pass a set of identities of plug-in modules 191 that identify only those plug-in modules 160 listed in the list of services 170. Thereafter, in a process which will be explained shortly, the plug-in initializer 152 can initiate and query that set of plug-in modules 160 in order to determine any other plug-in modules which may be required for services. Thereafter, the plug-in initializer 152 can then access those additional plug-in module definitions 162 for those additional plug-in modules 160 and can initialize and query those plug-in is as well. This process may be repeated until all plug-in modules 160 have been initiated and queries such that there are no remaining required but uninitiated plug-in modules.

Generally, the plug-in initializer 152 is responsible for retrieving or deriving a plug-in dependency list 192 indicating respective plug-in services provided by, and required by, each plug-in module 160 identified in the identities of plug-in modules 191 (i.e., Step 203 in Figure 2).

To do so in this example embodiment, in step 304, the plug-in initializer 152 enters a loop that is performed for each plug-in module 160 identified in the identities of plug-in modules 191 (as received from the plug-in module identifier 151).

In step 305 within the loop defined by step 304, the plug-in initializer 152 obtains plug-in initiation information 164 corresponding to the plug-in module definition 162 associated with the identity of the plug-in module 160 which is being processed in this iteration of the loop defined by step 304. The plug-in initiation information 164 can include, for example, previously saved plug-in state information produced as a result of a former operation of the plug-in module 160. In addition, plug-in initiation information 164 can include any properties, parameters or other signaling information that must be

provided to a particular plug-in module 160 when that plug-in module is called by the plug-in manager process 150 for operation (e.g., execution, interpretation or performance). Since the plug-in initiation information 164 can include previously saved state information 164, this allows a plug-in module 160 to begin operation at an execution point at which it left off during a former operation. In other words, if a plug-in module 160 saves state information when it exits or terminate operations, it may use this state information to continue or resume processing where left off since the plug-in manager of embodiment of this invention is able to save this state and is able to return it to the plug-in module 160 upon its next operation.

Next, in step 306, the plug-in initializer 152 instantiates the plug-in module 160 based upon the plug-in module definition 162 associated with the identity of the plug-in module being processed in this iteration of the loop defined by step 304. The plug-in module definition 162 may be, for example, a Java class which the plug-in initializer 152 instantiates within the memory 122 in the server computer system 120.

In step 307, the plug-in initializer 152 passes the plug-in initiation information 164 to the instantiated plug-in module 160. In this manner, the plug-in module 160 receives any required parameters, control commands, signaling information or plug-in state information in order to properly operate in an enabled state within the server computer system 120. At this point, according to this embodiment of the invention, the plug-in module 160 operates in an “enabled state” and is not yet providing a service 147.

In step 308, the plug-in initializer 152 queries a dependency interface associated with the plug-in module 160 using a dependency query 171 to obtain a dependency response 173 from the plug-in module 160. In the example of plug-in module architecture illustrated in Figure 4, the dependency interface is the service interfaces 161 for each respective plug-in service that the plug-in module 160 provides, as well as the list of required services 162 that the plug-in module 160 requires to properly operate. In other words, in step 308, the plug-in initializer 152 accesses the service interfaces 161 and required services 162 within the plug-in module 160 to determine which services are provided by, and required by, the plug-in module 160.

Next, in step 309, the plug-in module receives a dependency response 173 from the plug-in module 160. The dependency response 173 indicates respective plug-in services provided by, and required by, the plug-in module 160.

In step 310, the plug-in initializer 152 stores the identities of respective plug-in services provided by, and required by, the plug-in module 160, as identified in the dependency response 173, within a dependency list 192.

After all iterations of steps 305 through 310 are complete, the plug-in initializer 152 has instantiated a respective plug-in module 160 associated with each service that is to operate within the server computer system 120. In addition, the plug-in initializer 152 has queried each plug-in module 160 to retrieve, develop or compile a complete dependency list 192 of all services required and provided by those plug-in modules 160. Thereafter, the plug-in initializer 152 passes the dependency list 192 to the dependency analyzer 153.

Generally, the dependency analyzer 153 calculates a plug-in initiation order 190 based on the dependency list 192 (step 204 in Figure 2).

To do so, in step 311, the dependency analyzer 153 arranges a placement of each plug-in module 160 identified in the dependency list 192 within a plug-in initiation order 190 such that plug-in modules 160 not requiring services provided by other plug-in modules 160 are placed earlier in the initiation order 190, and such that plug-in modules 160 requiring services provided by other plug-in modules are placed later in the initiation order 190. In the example embodiment in Figure 5, the initiation order 190 is a plug-in module dependency tree which will be explained shortly. Generally, the plug-in module dependency tree 190 identifies or conveys a start or initiation order for all plug-in modules 160 identified by the plug-in module identifier 151.

In step 312, the dependency analyzer 153 analyzes the dependency list 192 indicating respective plug-in services provided by, and required by, each plug-in module 160 in order to determine which plug-in modules 160 provide services relied upon by other modules. In other words, in step 312, the dependency analyzer 153 determines a start order in which plug-in modules 160 can be transitioned from an enabled state into a service operation or started state such that when a particular plug-in module 160 enters its

respective service operation state, all services 147 required by that plug-in module 160 (e.g., services provided by other plug-ins) will be available at that point in time.

Next, in step 313, the dependency analyzer 153 creates at least one plug-in module dependency tree 190 (i.e., the plug-in initiation order) based on the step of analyzing (step 312). The plug-in module dependency tree 190 contains a hierarchical arrangement of nodes associated with respective plug-in modules 160. The hierarchical arrangement indicates the plug-in initiation order of plug-in modules 160 respectively associated with the nodes in the dependency tree 190.

Figure 8 illustrates an example of a set of plug-in module dependency trees 190-1 and 190-2 that contain plug-in module dependencies for the set of plug-in modules 160 illustrated in Figure 5. Each node in a plug-in module dependency tree 190 represents a particular plug-in service and the lower a node is in the tree hierarchy, the sooner or earlier the service associated with that node is to be initiated in relation to other services represented by other nodes in the plug-in module dependency tree 190. In other words, the plug-in manager 150 can traverse a plug-in module tree 190 to initiate plug-in modules 160 corresponding to the lowest nodes in the tree 190 first, before initiating service operation of plug-in modules 160 corresponding to upper layer nodes in the tree 190.

In particular, the plug-in module dependency tree 190-1 includes nodes 401 to 405 which each represent a respective service provided by a respective plug-in module 160. As illustrated, the root node 401 represents plug-in module 160-1 which provides service "A." As indicated in the plug-in modules 160 in Figure 5, service "A" in plug-in module 160-1 requires or depends upon services "B" and "E." As such, the plug-in module dependency tree 190-1 contains child nodes 402 in 403 of node 401 which represent plug-in modules 160-2 and 160-5 for services "B" and "E" respectively. Since the nodes 402 and 403 exist below node 401 (i.e., are children of node 401, which is a parent node), the plug-in manager 150 should initiate the services "B" and "E" before service "A" associated with node 401. Likewise, node 402 representing service "B" depends upon services "D" and "C", which are represented by nodes 404 and 405, respectively. Node 403 representing plug-in module service "E" requires or depends



upon service “C” as represented by node 405 in the plug-in module dependency tree 190-1.

With respect to plug-in module dependency tree 190-2, the node 406 represents plug-in module 160-8 which provides plug-in service “H.1.” This service “H.1” depends upon the service “G.1” of plug-in module 160-7 represented by node 407 in a tree 190-2. In addition, the service “G.1” represented by node 407 requires or depends upon the service “H.1” represented by node 406. As such, a service interdependency exists in the plug-in module dependency tree 190-2 which is represented by two dependency arrows 198-1 and 198-2 extending between the nodes 406 and 407. Also as indicated in the plug-in module dependency tree 190-2, the service “G.1” further depends upon the operation of the service “D” provided by plug-in module 160-4 as represented by node 408 in the plug-in module dependency tree 190-2.

Returning attention now to the flow chart of processing steps in Figure 7, in step 314, the service controller 154 traverses (e.g., via a bottom-up traversal) each plug-in module dependency tree 190 according to the hierarchical arrangement of nodes and, for each node encountered in the traversal, initiates service operation of the respective plug-in module 160 associated with that node.

Specifically, in step 315, the service controller 154 determines, from a published list of services 180 available for initiated plug-in modules 160, an identity of each initiated plug-in service 147 (or 162-5 for example in Figure 5) required by the respective plug-in module 160 being processed in this iteration of the tree traversal defined by step 314. In other words, in step 315, service controller 154 determines the identity of each service 147 required by the plug-in module 160 which is about to enter the service operation state.

Next, in step 316, the service controller 154 forwards, to the respective plug-in module 160 (being processed at this point in the traversal of the dependency tree 190), via the dependency available interface 163 associated with that respective plug-in module 160, the identity of each initiated plug-in service required by that respective plug-in module 160. In other words, in step 316, the service controller 154 provides the identities

of the required plug-in services to the plug-in module so that the plug-in module 160 can reference those required services during service operation .

When the plug-in module 160 receives the identities of the required services via the dependency available interface 163, the plug-in module 160 transitions to the started or service providing state and provides back to the plug-in manager 150 a list of services 174 (Figures 1 and 5) initiated within that plug-in module 160. In other words, when a plug-in module 160 enters the started state it returns, back to the plug-in manager 150, a list 174 of the services 147 which that plug-in module 160 is now providing for use by applications such as the server process 145.

Accordingly, in step 317, the service controller 154 receives a list of services 174 initiated by the respective plug-in module 160.

Next, the service controller 154 adds the list of services 193 provided by the respective plug-in module 160 to the published list of services 180. In this manner, the service controller 154 maintains or updates the current published list of services with any newly available services 147 which are now offered by the plug-in module 160 which recently begin operation in a service providing state.

This process defined by the traversal of the plug-in module dependency tree(s) 190 (the tree traversal processing of step 314) continues until all plug-in modules 160 are initiated into a started state in the order defined by the plug-in module dependency tree 190. In this manner, services of plug-in modules are 160 started in a proper order to avoid dependency conflict or deadlock situations in which one plug-in module 160 depends upon services 147 of another which is not yet started.

As noted above, if two plug-in modules happened to depend upon each other, each may be started at approximately the same time in which case the service controller 154 can initiate the wait state operation 165 within one or more of the service interdependent plug-in modules 160 in order to allow those plug-in modules to intercommunicate with each other to agree on a proper service providing sequence.

Those skilled in the art will understand that there can be many other variations made to the operations of the embodiments explained above while still achieving the

same objectives of the invention. Such variations are intended to be covered by the scope of this invention.

As an example, the plug-in manager process 150 can operate in a multi-threaded mode in which a processing thread is provided for each plug-in 160 that is determined to be required. In such an embodiment, once the plug-in manager 150 obtains the identities of plug-in modules for services required, the plug-in manager process 150 can start a processing thread for each plug-in (e.g., one thread per plug-in, or one thread for a subset of plug-ins). The processing thread for each plug-in(s) can then query the plug-in(s) to determine what services are required by (if any), and provided by the plug-in(s) associated with that thread.

If the thread for a plug-in determines that no services are required by that plug-in, then that thread of the plug-in manager process can initiate service operation of its respective plug-in at that point in time. Once initiated, the plug-in or its thread can then add the service(s) now available from that plug-in to a list of available plug-in services. Other threads that are operating on behalf of other plug-ins can periodically query the list of available plug-in services in order to determine when services become available that their respective plug-ins require. When a thread for a plug-in detects that all services are available that that thread's plug-in requires, that thread can initiate service operation of its respective plug-in and the plug-in or the thread can then also add the service(s) now available from that plug-in to the list of available services, such that the list of services grows as plug-ins are initiated in this multi-threaded manner.

Each thread thus initiates service operation of its respective plug-in(s) in an independent manner, without a requirement to develop a dependency tree. In this embodiment, the "plug-in initiation order" is thus a collective determination made by all operating threads as to when to launch the service(s) provided by the plug-in(s) associated with each thread. Thus, with respect to the flow chart in Figure 2, the step of retrieving the dependency list (step 203) is done for each plug-in by its respective thread within the plug-in manager process, whereas the step of calculating a plug-in initiation order is done by the collective operation of all threads (i.e., each thread contributing to

the calculation by determining when its respective plug-in(s) can start based on other services that may or may not be required by that plug-in(s)).

The foregoing description of embodiments of the invention are not intended to be limiting. Rather, any limitations to embodiments of the invention are presented in the

5 following claims.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207  
2208  
2209  
2210